МІНІСТЕРСТВО ОСВІТИ І НАУКИ,
МОЛОДІ ТА СПОРТУ УКРАЇНИ

Національний технічний університет
«Харківський політехнічний інститут»

МЕТОДИЧНІ ВКАЗІВКИ
до лабораторної роботи
«Створення та налаштування програм у візуальному середовищі Delphi 2009»

з курсу «Програмування»
для студентів напряму 6.040302 – Інформатика
(спеціалізація «Соціальна інформатика»)

Затверджено редакційно-видавничою
радою університету,
протокол № 2 від 01.12.10.

Харків НТУ «ХПІ» 2011

Методичні вказівки до лабораторної роботи «Створення та налаштування програм у візуальному середовищі Delphi 2009» з курсу «Програмування» для студентів напряму 6.040302 – Інформатика (спеціалізація «Соціальна інформатика») / Уклад. М. І. Безменов. – Х. : НТУ «ХПІ», 2011. – 37 с.

Укладач М. І. Безменов

Рецензент Л. М. Любчик

Кафедра системного аналізу і управління

ВСТУП

Сучасне програмування орієнтоване не написання програм на папері, а розробку їх у середовищі програмування з налаштунням у діалоговому режимі. У зв'язку із цим знання тільки безпосередньо мови програмування зараз є недостатнім – програміст повинен знати середовище програмування, уміти користуватися тими сервісними засобами, які надає це середовище.

Метою даної лабораторної роботи є освоєння інтегрованого середовища розроблювача (ICP) Code Gear Delphi 2009 і отримання початкових навичок у проектуванні Windows-застосунків з використанням компонентів Delphi та у налаштуванні цих застосунків.

1. ТЕОРЕТИЧНІ ОСНОВИ

1.1. Початкові дії

Після завантаження інтегрованого середовища розроблювача потрібно здійснити перехід до розробки нового застосунка, для чого можна у головному меню вибрати опцію File, у якій треба вибрати підопцію New і далі підопцію VCL Forms Application – Delphi, наслідком чого буде завантаження вікон Delphi 2009 для роботи з новим проектом (рис. 1.1).

На рис. 1.1 цифрами позначенно: *1* – головне вікно; *2* – вікно структури; *3* – вікно форми; *4* – вікно дизайнера форми; *5* – вікно інспектора об'єктів; *6* – вікно менеджера проектів; 7 – вікно палітри інструменту; *8* – закладка вікна коду; *9* – закладка вікна дизайнера форми; *10* – закладка вікна менеджера історії.

Щоб почати проектування нового застосунка можна також виконати таку послідовність дій:

1. Вибрати у головному меню опцію File, наслідком чого буде розкриття підменю, першою опцією якого є опція New.
2. У розкритому меню вибрати опцію New, в результаті чого відкривається підменю, однією з опцій якого є опція Other… (Інший…).
3. Вибрати опцію Other..., що приведе до відкриття вікна New Items (див. рис. 1.2).
4. У вікні New Items вибрати пункт VCL Forms Application, наслідком чого і буде відкриття вікон для проектування застосунка (рис. 1.1).

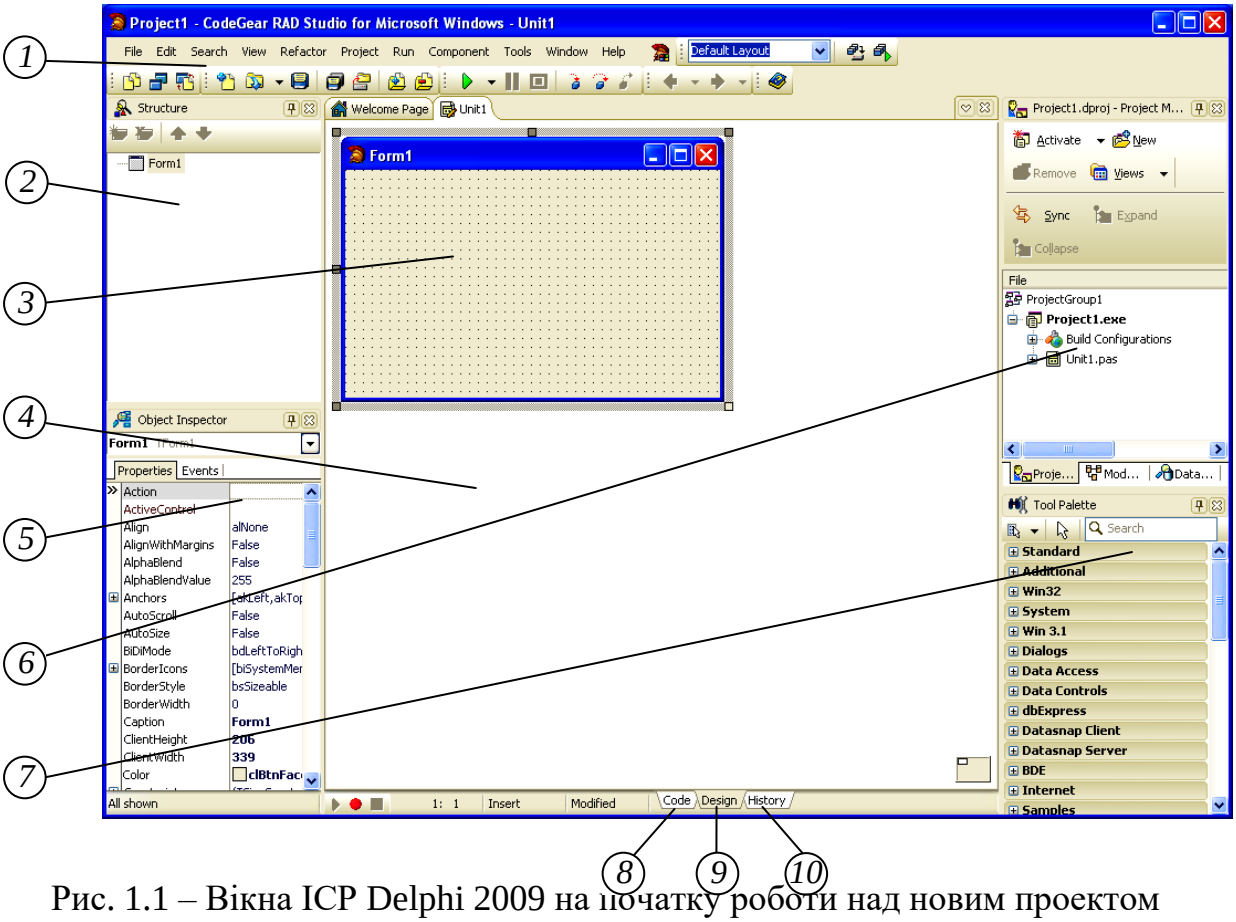


Рис. 1.1 – Вікна ICP Delphi 2009 на початку роботи над новим проектом

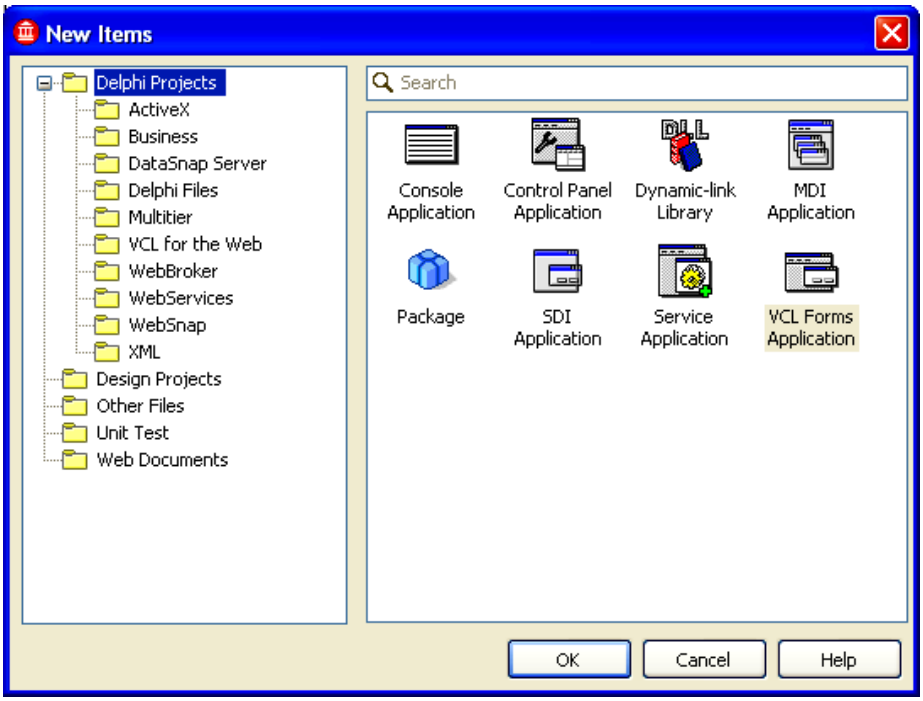


Рис. 1.2 – Вікно вибору виду застосунка, що проектуватиметься

Можна одразу увійти у вікно New Items, що відкривається за командою File ►New. Для цього треба клікнути мишкою над інструментальною кнопкою  (New Items).

1.2. Початкові відомості про середовище розроблювача Delphi 2009

1.2.1. Головне вікно

Це вікно містить у собі всі засоби з керування проектом (рис. 1.3), а саме, головне меню та набір інструментальних кнопок.

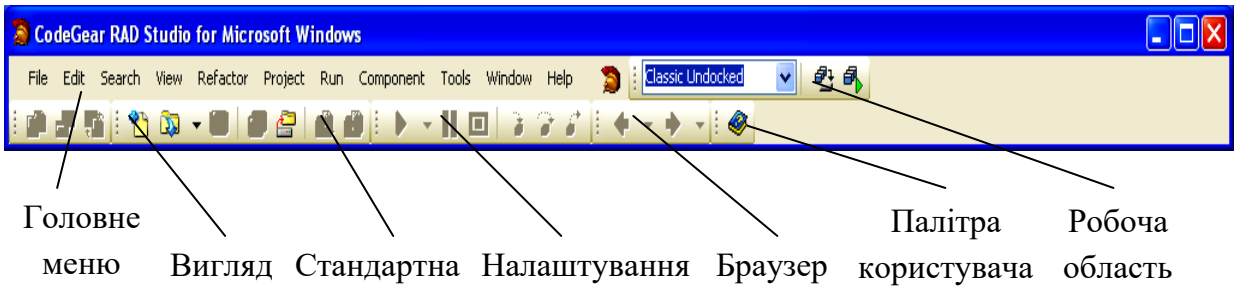


Рис. 1.3 – Основні панелі головного вікна ICP Delphi

За допомогою головного меню здійснюється керування можливостями ICP. Швидкий доступ до багатьох команд головного меню забезпечують так звані інструментальні кнопки, згруповані за функціональним призначенням на окремих панелях головного вікна. Панель Стандартна містить інструментальні кнопки, що забезпечують дублювання тих дій, які можна виконати, звернувшись до пунктів File (Файл) і Project (Проект) головного меню. Панель Вигляд поєднує кнопки, що дублюють деякі підпункти пункту View (Вигляд) головного меню, а панель Налаштування полегшує керування процесом налаштування та виконання програми, частково дублюючи підпункти пункту Run (Виконати) головного меню. За допомогою кнопок, що розміщені на панелі Браузер можна, можна переходити до сторінок, з якими здійснювалася робота при звертанні до Internet-ресурсів. Панель Робоча область дозволяє здійснювати керування виглядом робочої області ICP Delphi (за умовчанням, для режиму налаштування, класичний), а також збереження поточного її вигляду (за допомогою інструментальної кнопки ). Панель Палітра користува-

ча за умовчанням вміщує тільки одну інструментальну кнопку, що забезпечує доступ до довідкової системи Delphi.

1.2.2. Вікно дизайнера форми

На початку роботи над новим проектом вікно дизайнера форми у вигляді окремої сторінки із закладкою Design розташовується над сторінкою вітання, за умовчанням будучи відкритим. У правому нижньому куті сторінки дизайнера форми виводиться невеликий затінений прямокутник, який умовно відображає екран при роботі застосунка. На зображенні екрана відображається місце розташування і розміри форми відразу ж після запуску застосунка.

Вікно форми (конструктор форми) виводиться в лівому верхньому куті вікна дизайнера форми і являє собою проект вікна створюваної програми. Спочатку воно містить кнопки виклику системного меню, розгортання, згортання і закриття вікна, рядок заголовка і габаритну рамку (тобто стандартні інтерфейсні елементи Windows), а також порожню робочу область.

Будучи візуальним засобом розміщення компонентів на формі, дизайнер форми дозволяє безпосередньо за допомогою мишки розміщати компоненти на формі з використанням Палітри Інструменту (Tool Palette), Інспектора Об'єктів (Object Inspector) або Дерева Об'єктів (Object Tree). При цьому необхідні зміни у вікні форми можуть бути внесені в будь-який момент під час проектування (наприклад, за допомогою мишки можна змінювати положення і розміри компонентів).

З формою, крім дизайнера форми, пов’язані ще два вікна, розташовані на окремих сторінках разом зі сторінкою дизайнера, а саме: вікно коду (закладка з ім’ям Code) і вікно менеджера історії (закладка з ім’ям History).

1.2.3. Вікно коду

У вікні коду програми відображається і редагується текст програми (найчастіше текст модуля), написаної мовою Delphi. При розробці нового проекту це вікно містить створюваний автоматично мінімальний початковий текст модуля. Цей текст забезпечує нормальне функціонування порожньої форми. Далі програміст модифікує даний текст відповідно до розв’язуваної задачі, причому зміни виконуються як за допомогою засобів бібліотеки

візуальних компонентів, так і за допомогою «ручної» корекції. При цьому досить часто доводиться переходити з вікна коду у вікно форми і зворотно.

Доступ до вікна коду (рис. 1.4) здійснюється або натисканням клавіші F12, або кліком мишкою над інструментальною кнопкою  на панелі Вигляд ICP, або виконанням команди View ► Toggle Form/Unit. Зворотний перехід у вікно дизайнера форми здійснюється аналогічно. Можна також виконувати кліки мишою над закладками Code і Design.

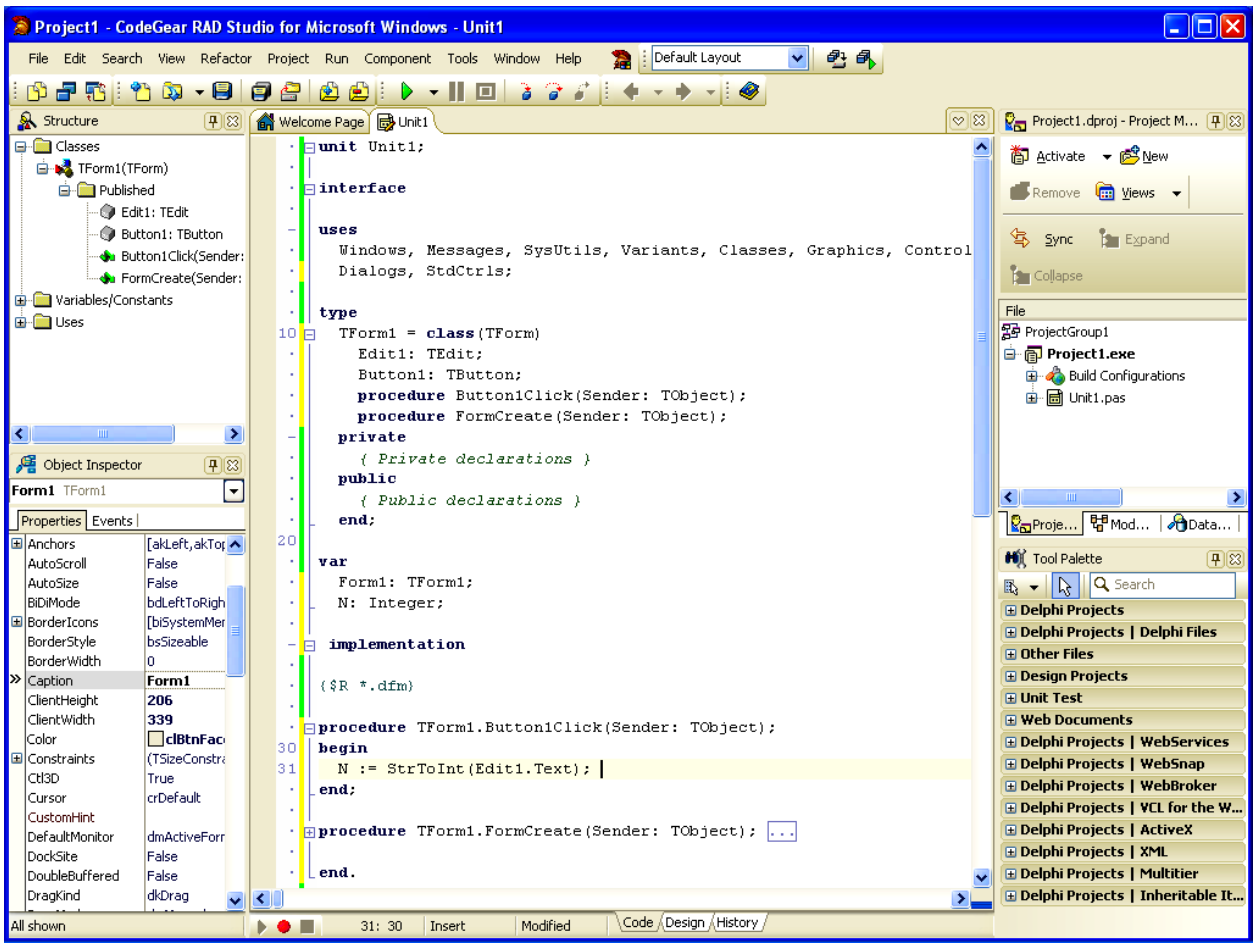


Рис. 1.4 – Вигляд ICP Delphi 2009 з відкритим вікном коду

Якщо програма є багатомодульною, кожен модуль може бути завантажений на окремій сторінці вікна коду, причому кожна сторінка має закладку з іменем модуля (на рис. 1.4 завантажений тільки один файл).

1.2.4. Менеджер історії

Менеджер історії (History Manager) розташовується на одній із вкладок вікна дизайнера форми, як і вікно коду. Доступ до нього здійснюється кліком

мишкою над закладкою History у нижній частині вікна дизайнера форми. Менеджер історії дозволяє побачити і порівняти різні версії файлу, включаючи резервні копії. Використовуючи менеджер історії, можна, зокрема, повернутися до необхідної версії файлу, причому це можна зробити у будь-який час.

1.2.5. Вікно інспектора об'єктів

Вікно Інспектора Об'єктів (Object Inspector) призначене для зміни параметрів (характеристик) розміщених на формі компонентів. За допомогою мишки у вікні форми можна коректувати тільки частину параметрів, які характеризують компоненти, що розміщені на формі. Користуючись же Інспектором Об'єктів, програміст може коректувати всі характеристики компонентів, які доступні на етапі проектування програми (наприклад, колір, шрифт і текст напису, колір компонента). Для здійснення цих змін в Інспекторі Об'єктів використовуються дві вкладки (сторінки) – Properties (Властивості) та Events (Події). За допомогою вкладки Properties (Властивості) програміст може змінювати властивості компонента, а за допомогою вкладки Events (Події) – призначати реакцію компонента на ту або іншу подію (реакцію на натискання клавіш, клік мишкою, зміну розміру вікна, появу компонента на екрані тощо).

Для зміни властивостей компонента необхідно перейти в Інспекторі Об'єктів на першу вкладку, що має вигляд таблиці, у першому стовпчику якої містяться імена властивостей. Значення властивості відображається поруч з її ім'ям у другому стовпчику.

Властивості компонентів можуть відображатися єдиним значенням (числом, рядком символів, True – Істина чи False – Неправда) або множиною значень. У першому випадку говорять про прості властивості, а в другому – про складні. Ліворуч від імені складної властивості відображається значок ⊞.

Для задавання значення простої властивості необхідно вибрати відповідний рядок і ввести потрібне значення в правому стовпчику. Якщо при виборі простої властивості в правому кінці рядка з'являється кнопка ▾, то це означає, що для даної властивості визначений список можливих значень, який розкривається при кліку над кнопкою ▾ і служить для подальшого вибору потрібного значення. Для зміни значення складної властивості слід клацнути мишкою над значком ⊞ поруч з його ім'ям, у результаті чого відкривається список складових цієї властивості. Закриття цього списку

здійснюється кліком мишкою над значком  , у який перетворюється значок  при відкритті списку.

Наприкінці рядка для деяких із властивостей при їх активізації може з'явитися кнопка  . Клік над цією кнопкою приводить до появи на екрані діалогового вікна, що служить для установки значення властивості.

Вкладка Events (Події) також має вигляд таблиці з декількох рядків і двох стовпчиків. У першому стовпчику відображається ім'я події, а в другому – ім'я підпрограми для її опрацювання (ім'я опрацьовувача події).

Крім двох вкладок, у верхній частині вікна Інспектора Об'єктів є список, який може розкриватися і містить імена всіх поміщених на форму компонентів із зазначенням їхніх класів. При кліку мишкою над ім'ям компонента в цьому списку відбувається переключення на відповідні таблиці в Інспекторі Об'єктів, а також активізація обраного компонента у вікні форми, що позначається виділенням компонента прямокутником.

Повторне відкриття раніше закритого вікна Інспектора Об'єктів виконується командою View ► Object Inspector (клавіша Alt+Enter).

1.2.6. Вікно структури

Вміст вікна структури (Structure) залежить від того, що завантажено в центральне вікно – дизайнер форми або вікно коду. Якщо в центральному вікні відкритий дизайнер форми, то у вікні структури міститься Дерево Об'єктів (Object TreeView), а при відкритому вікні коду у вікно структури завантажується браузер коду.

Дерево Об'єктів наочно відображає зв'язки між окремими візуальними та невізуальними компонентами, розміщеними на активній формі або в активному модулі даних, а саме, приналежність одних компонентів іншим. Окремі компоненти відображаються в Дереві Об'єктів за допомогою піктограм (маленьких рисунків), поруч із якими містяться імена відповідних компонентів. Клацання (клік) лівою кнопкою мишки на кожній з таких піктограм приводить до активізації відповідного компонента у вікні форми і відображення його властивостей у вікні Інспектора Об'єктів. При подвійному кліку на піктограмі компонента спрацьовує так званий механізм Code Insight, що вставляє у вікно коду заготівку для опрацьовувача події OnClick (За кліком) або опрацьовувача іншої події (залежно від типу компонента). При цьому

автоматично відкривається вікно коду, а курсор розміщується усередині заготовки опрацьовувача події.

Використовуючи Дерево Об'єктів, можна змінити власника компонента, для чого досить клацнути лівою кнопкою мишки над піктограмою і, не відпускаючи кнопку мишки, «перетягнути» її у вікні на піктограму нового власника. Натискання клавіші Delete приводить до видалення компонента, який виділений у Дереві Об'єктів.

Вікно Браузера Коду (Code Explorer) звичайно активізується разом з вікном коду і служить для полегшення пошуку потрібних елементів у великому тексті програми. Це вікно містить елементи, що використовуються у завантаженому у вікно коду тексті.

Закрити вікно структури можна, клікнувши мишкою над кнопкою  в його правому верхньому куті. Повторне відкриття раніше закритого вікна структури виконується командою View ► Structure.

1.2.7. Вікно палітри інструменту

Проектування програми у ICP Delphi пратично неможливе без використання палітри інструменту, яка забезпечує додавання компонентів до активної форми при проектуванні інтерфейсу програми. За умовчанням в ICP Delphi 2009 палітра інструменту розміщена в спеціальному вікні Tool Palette, яке розташоване праворуч від вікна дизайнера форми. Окремі вкладки палітри інструменту розміщені у вікні у вигляді списку вкладок, кожна з яких розкривається/закривається кліком мишкою над значком або ліворуч від назви вкладки (рис. 1.5, *а*, *б*).

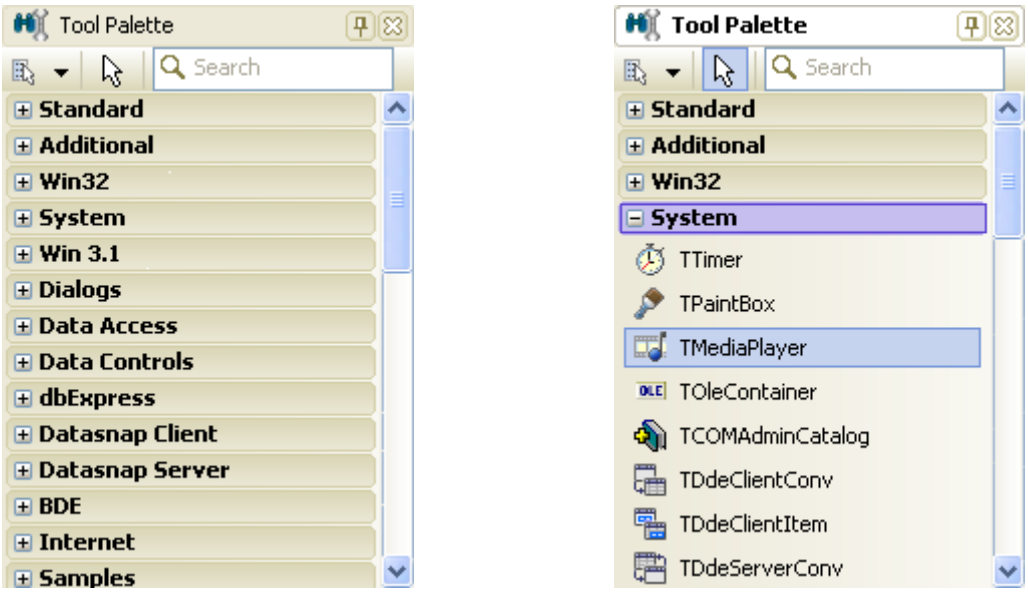
Компоненти при розкритих вкладках відображаються піктограмами, поруч із кожною з яких зазначається клас відповідного компонента (рис. 1.5, *б*). Щоб розмістити компонент на формі, треба вибрати компонент у палітрі інструменту, здійснивши над ним клік мишкою, після чого треба клікнути мишкою над зображенням форми у вікні дизайнера форми. Компоненти можна розміщати на формі не тільки за кліком мишкою, але й перетаскувати з палітри інструменту при натиснутій лівій кнопці мишки.

1.2.8. Вікно менеджера проектів

Вікно менеджера проектів (Project Manger) розташоване над вікном палітри інструменту і служить для відображення структури проекту, над яким

веде роботу розроблювач застосунка. За допомогою менеджера проектів можна додавати до проекту файли (модулі), видаляти їх і перейменовувати. Можна також комбінувати проекти, формуючи зв'язану групу проектів.

Повторне відкриття раніше закритого вікна менеджера проектів виконується командою View ► Project Manger (Alt+0).



а *б*

Рис. 1.5 – Вікно палітри інструменту при відкритому вікні форми:
а – з закритими вкладками; *б* – з відкритою вкладкою System

1.3. Проект

Мінімальний код, що може нормально функціонувати під керуванням операційної системи, створюється автоматично відразу ж після вибору команди File ► New ► VCL Forms Application – Delphi (Файл ► Створити ► Застосунок з VCL формами).

Якщо говорити про текст програми, то як такий можна розглядати так званий файл проекту, що має розширення .DPR. Саме цей файл обробляється компілятором, будучи головною програмною одиницею, до якої підключаються одна або кілька інших програмних одиниць, що називаються модулями та зберігаються у файлах з розширенням .PAS.

Стандартний текст проекту для функціонування форми у разі проектування застосунка у середовищі програмування Delphi 2009 має такий вигляд:

```
program Project1;

uses
  Forms,
  Unit1 in 'Unit1.pas' {Form1};

{$R *.res}

begin
  Application.Initialize;
  Application.MainFormOnTaskbar := True;
  Application.CreateForm(TForm1, Form1);
  Application.Run;
end.
```

Зазвичай (якщо не проводилося переналаштування) у вікні коду автоматично виділяються жирним шрифтом так звані зарезервовані слова і курсивом – коментарі.

Файл проекту висвітлюється у вікні коду або на запит програміста, або з появою деяких помилок у ході виконання програми. Звичайно у вікні коду він не відображається і змінюється тільки достаньо кваліфікированим програмістом, і то досить рідко.

1.4. Модуль форми

Модулі – це програмні одиниці, що служать для розміщення окремих частин програм.

Модулем форми є модуль, до якого входять елементи програмного коду, що забезпечують створення та функціонування форми та тих компонентів, які розміщуються на ній.

На вкладці Unit вікна коду у випадку стандартної форми міститься код, сформований Delphi, що в обов'язковому порядку повинен бути змінений програмістом (рис. 1.6).

В інтерфейсній секції наведеного програмного коду підключені стандартні модулі, а також описані один тип (клас `TForm1`) та один об'єкт (змінна `Form1`).

Якщо на етапі проектування програміст розміщає на формі новий компонент, то в текст модуля автоматично вставляється опис цього компонента та за додатковими командами створюються заготівки опрацьовувачів подій, на які реагує цей компонент. Справа програміста – наповнити опрацьовувачі

подій конкретним змістом у вигляді операторів. Програміст може оголошувати нові змінні, типи, константи і т. д., однак видаляти та змінювати рядки, вставлені автоматично, не рекомендується. Наприклад, якщо у тексті модуля є опрацьовувач деякої події і треба його видалити, то необхідно знищити тільки рядки, що вставлялися програмістом. При компіляції ж буде видалений увесь текст, що вставлявся автоматично.

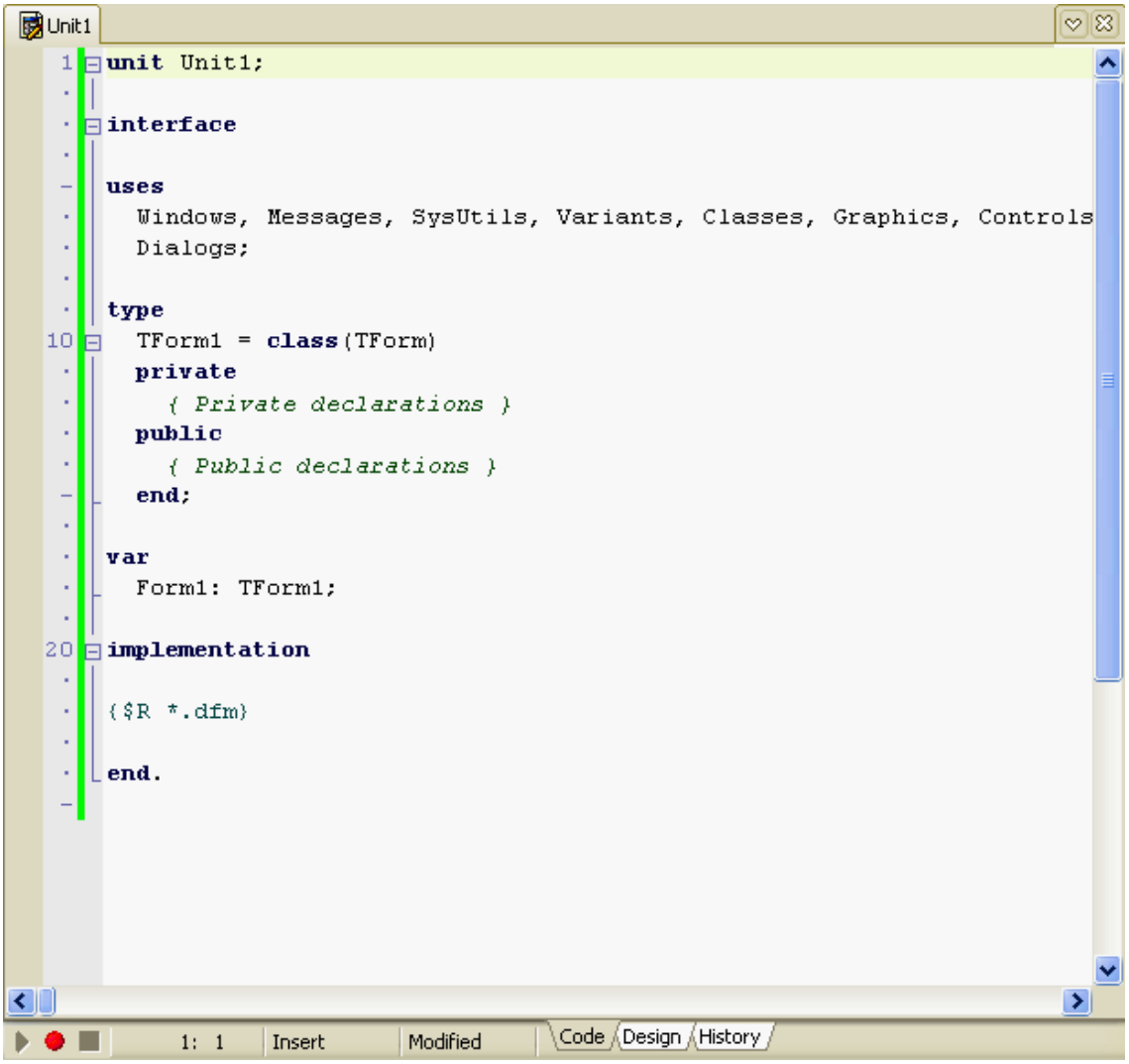


Рис. 1.6 – Початковий вигляд вікна коду

1.5. Налаштування програми

Процес усунення помилок у програмі називається ***налаштуванням*** (debugging), а самі помилки в програмі досить часто називають ***жучками*** (bugs). Серед помилок, які допускаються в програмі, виділяють синтаксичні помилки, помилки часу виконання і логічні помилки.

Синтаксичні помилки – це помилки, пов'язані з порушенням ***синтаксису*** (тобто правил граматики) мови програмування. Прикладом такої помилки, що зустрічається досить часто, є пропуск символу «крапка з комою», яким розділяються оператори у програмах, написаних мовою Delphi. При виявленні синтаксичної помилки компілятор видає ***повідомлення про помилку***, вказуючи її передбачуване місце розташування і пояснюючи можливий її зміст. Звісно, природа помилки може відрізнятися від тієї її інтерпретації, яку робить компілятор, тим більше що одна синтаксична помилка може призвести до того, що компілятор видасть декілька повідомлень про помилки, зумовлені, проте, наявністю лише першої з них. Може бути і протилежний випадок, коли одна синтаксична помилка ховає від компілятора інші помилки, що є далі у тексті програми.

Досить часто компілятор виводить ***натяки*** та ***попереджувальні повідомлення*** про деякі дещо незвичні конструкції у програмі. У багатьох випадках програмісти звертають увагу тільки на повідомлення компілятора про наявність помилок (Errors), не реагуючи на натяки (Hints) та попередження (Warnings). У той же час попереджувальні повідомлення можуть свідчити про помилки, у зв'язку з чим на них варто обов'язково звертати увагу і вносити відповідні виправлення. В ідеальному випадку попереджувальні повідомлення відсутні.

Для запуску процесу компіляції програми слід обрати пункт меню Project ► Compile Project (гаряча клавіша Ctrl+F7). Якщо створений програмний код є некоректним, то по закінченню процесу компіляції на екрані буде відображене вікно (див. рис. 1.7), в якому міститься сумарна інформація про виявлені в процесі компіляції помилки (Errors), натяки (Hints), та попередження (Warnings).

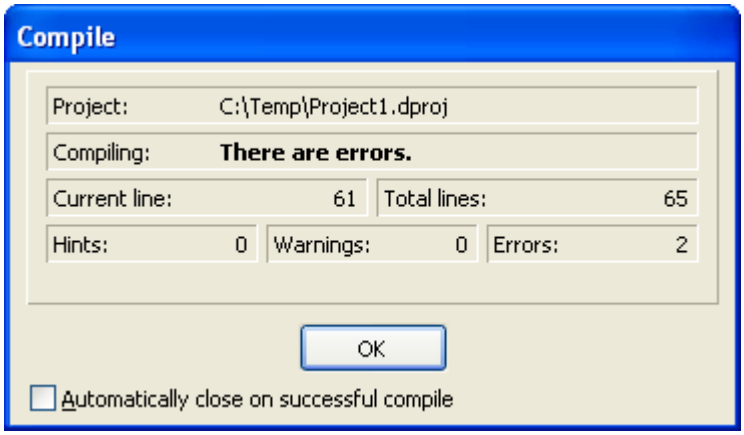


Рис. 1.7 – Вікно компіляції проекту у разі наявності помилок

Якщо здійснювалася тільки компіляція і у проекті відсутні помилки, то кінцевий вигляд вікна компіляції буде аналогічний зображеному на рис. 1.8.

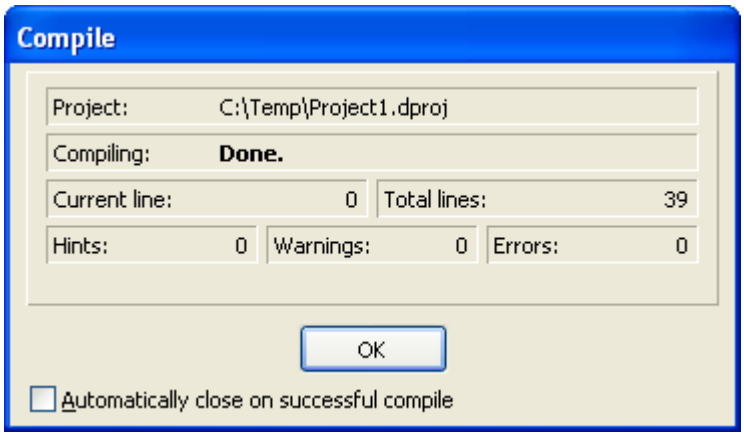


Рис. 1.8 – Вікно компіляції проекту у разі відсутності помилок

Можливим є також вибір пункту меню Run ►Run (гаряча клавіша F5 або клік мишкою над інструментальною кнопкою). В останньому випадку, насправді, мова йде про запуск програми на виконання. Але перед виконанням програми, якщо у її тексті здійснювалася корекція, будуть автоматично виконані компіляція та редагування зв'язків. У разі відсутності синтаксичних помилок та помилок у редагуванні зв'язків програма буде автоматично запущена на виконання, інакше їх необхідно усунути.

Можливий також безпосередній запуск редактора зв'язків після виконання компіляції або ж одразу. Для цього потрібно виконати команду Project ► Build Project (гаряча клавіша F7). При нормальному завершенні роботи редактора зв'язків на екран виводиться вікно, що має вигляд, зображений на рис. 1.9.

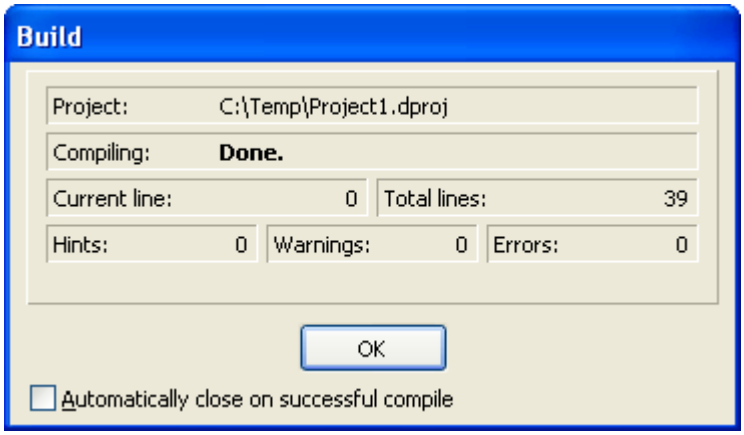


Рис. 1.9 – Вікно редагування зв'язків у разі відсутності помилок

Деякі помилки проявляються тільки під час виконання програми, у зв'язку з чим вони носять відповідну назву – ***помилки часу виконання*** (run-time errors). У термінології Delphi такі помилки називаються винятками (Exceptions). З появою таких помилок програма завершується аварійно з видачею пояснювального повідомлення. У середовищі поява таких помилок призводить до переривання виконання програми і виведення вікна з інформацією про вид виключення (рис. 1.10).

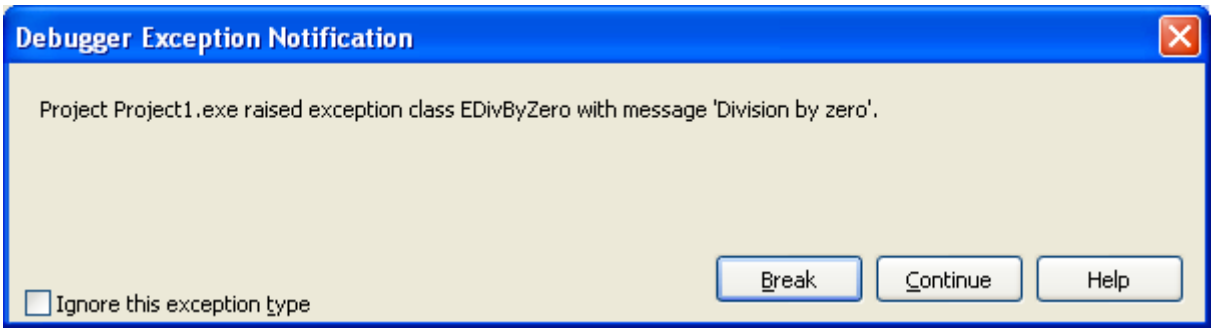


Рис. 1.10 – Вікно виникнення виключення

Прикладами таких помилок часу виконання є ділення на нуль, одержання великих числових значень, які не можуть бути записані в комірку пам'яті (переповнення).

Часто найбільш неприємними помилками є ***логічні помилки*** – помилки в самому алгоритмі, а також помилки, спричинені елементарною неуважністю (наприклад, використання в програмі операції * замість операції +, задавання неправильного числового значення, використання одного імені змінної замість іншого). Такі помилки компілятор найчастіше виявити не може (за винятком випадків, коли вони призводять до порушення синтаксису). Мало того, логічні помилки можуть не проявити себе і під час виконання програми як помилки часу виконання.

Для виявлення логічних помилок здійснюється ***тестування*** програми, яке виражається в її запуску з кількома характерними наборами вхідних даних і перевірці відповідних їм результатів. При цьому в жодному разі не можна обмежуватися одноразовою перевіркою програми – повинні бути відслідковані всі окремі випадки вхідних даних, з якими програма може зіткнутися. Тільки після перевірки правильності функціонування програми на багатьох характерних наборах даних можна отримати високу (але не абсолютну) гарантію її правильності.

Найчастіше за все, щоб знайти причину помилки, треба виконати якийсь фрагмент програми, спостерігаючи значення змінних при виконанні кожного оператора.

Для виявлення більшості логічних помилок передбачений спеціальний засіб, що зветься налаштувальником (Debugger).

Робота в режимі налаштування – це насамперед виконання програми по кроках з відслідковуванням послідовності виконання операторів програми та значень, які набувають змінні після виконання того або іншого кроку.

Для виконання фрагменту програми по кроках можна використовувати такі команди меню:

- Run ►Trace Into (Трасування із заходом у ...) – покрокове виконання програми із заходом у процедури та функції з наступним покроковим виконанням їх рядків (аналогом є натискання клавіші F11 або клік мишкою над інструментальною кнопкою );
- Run ►Step Over (По крокам без заходу у ...) – покрокове виконання рядків програми, при якому виклик пілпрограми вважається за одну команду, тобто вхід у неї не проводиться (аналогом є натискання клавіші F10 або клік мишкою над інструментальною кнопкою );
- Run ►Trace to Next Source Line (Трасування до наступного рядка) – перехід до наступного рядка програми (аналогом є натискання клавіші Shift+F7);
- Run ►Run to Cursor (Виконати до курсору) – команда виконує програму до того виконуваного оператора, на якому розташований курсор у вікні редактора коду (аналогом є натискання клавіші Ctrl+F10);
- Run Until Return (Виконати до виходу з функції) – виконання програми до виходу з поточної функції і зупинка на операторі, наступному за викликом цієї функції, з залишенням курсору у рядку, в якому було здійснене звертання до функції (аналогом є натискання клавіші Shift+F11 або клік мишкою над інструментальною кнопкою );
- Run ►Show Execution Point (Показати точку виконання) – команда показує на екрані виконуваний рядок коду та переміщує на нього курсор.

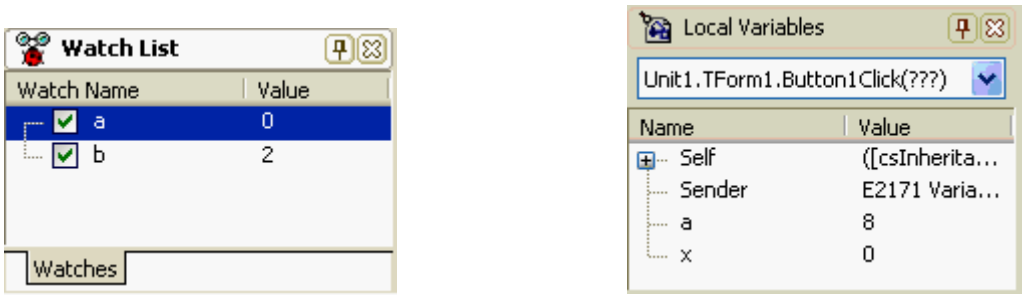
Досить часто для налаштування програми використовуються так звані точки переривання. Щоб ввести просту (безумовну) точку переривання, достатньо у вікні редактора коду клікнути мишкою на лівій межі вікна навпроти необхідної рядка коду. При цьому здійсниться забарвлення рядка, а

навпроти нього на лівій межі вікна коду з'явиться червона кулька. Якщо тепер запустити програму на виконання, то кожен раз коли керування перейде до рядка, в якому вказана точка переривання, відбувається переривання виконання. Відміна точки переривання здійснюється повторним кліком мишкою на лівій межі вікна коду навпроти рядка з точкою переривання. Точку переривання можна встановити командою Run ►Add Breakpoint ►Source Breakpoint з введенням номера рядка, де повинно здійснюватися переривання.

Дія точки переривання дещо аналогічна натисканню клавіші Ctrl+F10, але перевага точок переривання полягає в тому, що можна одночасно вказати декілька таких точок в різних місцях коду і в різних модулях. Програма в цьому разі виконується до першої точки переривання, яка зустрінеться під час виконання. Після аналізу проміжних результатів можна продовжити виконання програми, натиснувши інструментальну кнопку  .

Точки переривання можна встановлювати тільки на виконуваних операторах. Прибрати точку переривання можна, виконавши ті ж дії, що виконувалися для її встановлення.

Для перегляду значення змінних у момент зупинки програми досить навести курсор мишки на ім'я необхідної змінної в коді програми. Якщо ж є необхідність відслідковувати стан одразу декількох змінних, їх можна додати у вікно перегляду Watches List (див. рис. 1.11, *а*).



а *б*

Рис. 1.11 – Вікна перегляду стану змінних:
а – вікно перегляду змінних та виразів; *б* – вікно локальних змінних

За умовчанням вікно перегляду стану змінних автоматично виводиться у режимі налаштування ліворуч від вікна коду. У разі відсутності потреби у цьому вікні, його можна закрити. Щоб викликати раніше закрите вікно перегляду стану змінних, необхідно вибрати в головному меню опцію View ► Debug Windows ► Watches або ж натиснути комбінацію клавіш Alt+3.

Для додавання змінних у вікно Watches List достатньо здійснити подвійний клік мишкою в цьому вікні і ввести назву змінної в рядок Expression вікна, яке відкриється після кліку. Можна також, активізувавши вікно Watches List, натиснути клавішу Insert чи Ctrl+A або, клікнувши правою кнопкою мишки у цьому вікні, відкрити контекстне меню і вибрати в ньому пункт Add Watch. Для видалення імені з вікна Watches List треба активізувати його у вікні та натиснути клавішу Delete чи Ctrl+D або скористатися правою кнопкою мишки і вибрати відповідний пункт у контекстному меню, що відкривається за кліком.

За умовчанням у режимі налаштування ліворуч від вікна коду під вікном Watches List виводиться вікно локальних змінних Local Variables (див. рис. 1.11, *б*).

У цьому вікні виводяться значення локальних змінних процедур і функцій (у тому числі їх формальних параметрів). У разі закриття вікна локальних змінних, його повторне відкриття здійснюється вибором в головному меню опції View ► Debug Windows ► Local Variables або ж натисканням комбінації клавіш Alt+4.

Якщо вміст вікна локальних змінних визначається автоматично, то вміст вікна Watches List визначається програмістом.

2. ПРИКЛАД СТВОРЕННЯ НОВОГО ПРОЕКТУ ТА ЙОГО НАЛАШТУВАННЯ

Розглянемо можливу послідовність дій по налаштуванню програми на прикладі задачі розробки програми відшукання коренів квадратного рівняння

$$ax^2 + bx + c = 0,\ a \neq 0.$$

Далі окремі групи дій умовно розіб'ємо на етапи з метою їх структуризації.

Етап 1. Конструювання форми.

Завантаживши Delphi, виберемо команду File ► New ► VCL Forms Application – Delphi. Захоплюючи по черзі межі форми при натиснутій лівій кнопці мишки, підберемо зручні розміри форми.

Далі слід розмістити на формі компоненти відповідно до вимог задачі. Задача передбачає наявність засобів із забезпечення введення трьох чисел, виведення результатів обчислень та безпосередньої реалізації процесу обчис-

лень. Природна річ, уведення даних завжди повинне супроводжуватися змістовними повідомленнями-підказками.

Проектування форми будемо здійснювати, орієнтуючись на рис. 2.1.

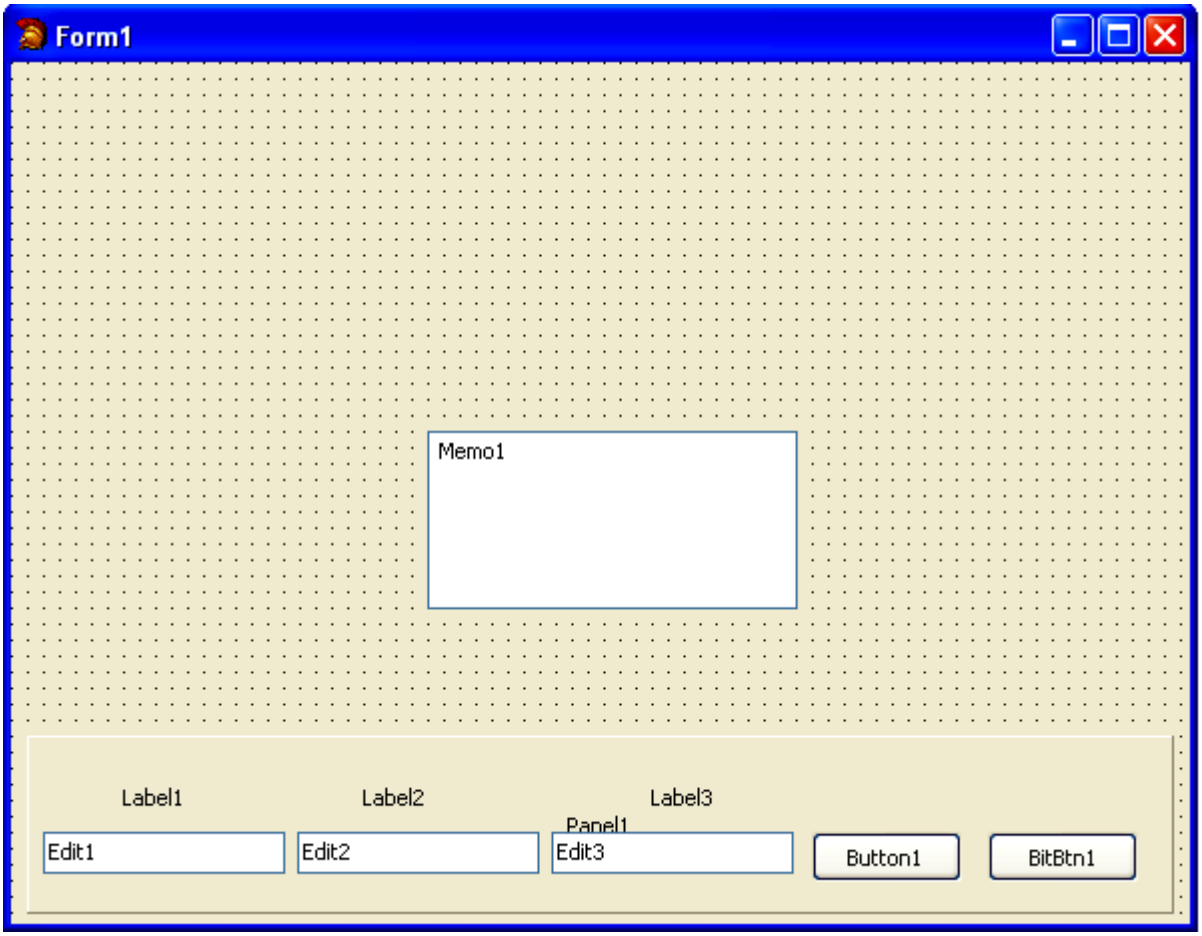


Рис. 2.1 – Можливий початковий вигляд форми

Виберемо на вкладці Standard палітри інструменту компонент TPanel (Панель) і клікнемо мишкою над формою. У результаті на формі з'явиться компонент TPanel (його лівий верхній кут розташується в точці, над якою здійснювався клік). Перетягнемо панель мишкою в нижню частину форми, розтягши її на всю ширину форми і задавши висоту, що дорівнює приблизно 0,2 від висоти форми (панель можна розташовувати в будь-якому місці форми). Вона буде служити контейнером, у якому розміститься інші компоненти проекту (насамперед вікна редакторів для введення даних і кнопки для подачі команди на початок обчислень та команди на завершення роботи програми). У принципі без панелі можна обійтися, розмістивши згадані елементи на вільних місцях форми, що є контейнером для всіх компонентів.

Аналогічним способом виберемо на вкладці Standard три компоненти TEdit, три компоненти TLabel та компонент TButton, а на вкладці Additional – компонент TBitBtn. Розмістимо ці компоненти на панелі (для чого будемо клікати мишкою над панеллю) і встановимо для них потрібні розміри і місце розташування.

Компонент TEdit (однорядкове редаговане текстове поле) застосовують для введення та (або) відображення досить довгих текстових рядків. Для зміни тексту, що міститься в ньому, достатньо змінити значення його властивості Text, що можна виконувати як на етапі проектування програми, так і програмними засобами. Ми будемо використовувати три таких компонент для введення трьох коефіцієнтів рівняння.

Компонент TLabel (Мітка) призначений для виведення різних повідомлень (підказок, результатів обчислень і т. д.). Ми будемо його використовувати для виведення підказок, що випереджають введення (значення якого з коефіцієнтів потрібно набрати у відповідному редакторі). Для зміни тексту, що міститься в цьому компоненті, досить змінити значення його властивості Caption, що також можна робити як на етапі проектування програми, так і програмними засобами.

Компонент TButton (Кнопка) призначений для керування програмою. Ми вдамося до нього для організації обчислень і забезпечення виведення.

Компонент TBitBtn (Кнопка з зображенням) є різновидом кнопки TButton. Ми застосуємо один з варіантів цього компонента, який служить для видачі сигналу про припинення роботи програми. Ця кнопка не є обов'язковою, оскільки припинення роботи програми забезпечується кліком над кнопкою ❎ закриття форми або натисканням сполучення клавіш Alt+F4. Замість цього компонента можна використовувати і звичайну кнопку TButton з відповідним опрацьовувачем події OnClick.

Нарешті, для виведення результату обчислень помістимо на форму багаторядкове редаговане текстове поле, для чого виберемо на вкладці Standard компонент TMemo (Текстове поле, від *memo field*) і клацнемо мишкою над вільним місцем форми (поза панеллю). Компонент TMemo вживають для уведення, редагування й (або) відображення досить довгих текстів, які можуть містити кілька рядків. Цей текст може корегуватися як на етапі візуального проектування, так і програмно, для чого необхідно змінювати властивість Lines (Рядки) цього компонента.

Відповідно до специфіки розв'язуваної задачі здійснимо такі зміни властивостей компонентів:

- Форма:
  Position — poScreenCenter
  Caption — Приклад
- Панель:
  Align — alBottom
  BevelOuter — bvNone
  Caption — очистимо
- Мітки Label1, Label2 та Label3:
  Caption — відповідно тексти Коефіцієнт a, Коефіцієнт b, Коефіцієнт c
- Редактори Edit1, Edit2 та Edit3:
  Text — 0
- Багаторядкове поле:
  Align — alClient
  Lines — очистимо
- Кнопка Button1:
  Caption — Обчислити
- Кнопка Close:
  Kind — bkClose

У результаті форма набуде вигляду, зображеного на рис. 2.2.

Розглянемо описані вище зміни.

Властивість Position (Положення) у форми визначає її положення щодо меж екрана. За умовчанням ця властивість має значення poDesigned й означає розташування вікна так, як воно було розміщене на етапі конструювання. Наведене значення poScreenCenter для властивості Position зумовлює розташування форми в центрі екрана.

Властивість Align (Вирівнювання) визначає спосіб розміщення даного компонента щодо того контейнера, який його містить. Наприклад, панель розташована усередині форми, а кнопки, мітки і поля введення – усередині панелі. Значення alBottom у цієї властивості, задане для панелі, забезпечує притиснення останньої до нижньої межі форми з розтягуванням її по всій довжині форми. У багаторядкового поля властивість Align дістала значення alClient, що «примушує» компонент зайняти всю частину форми, яка залишилася незаповненою (вона називається клієнтською областю).

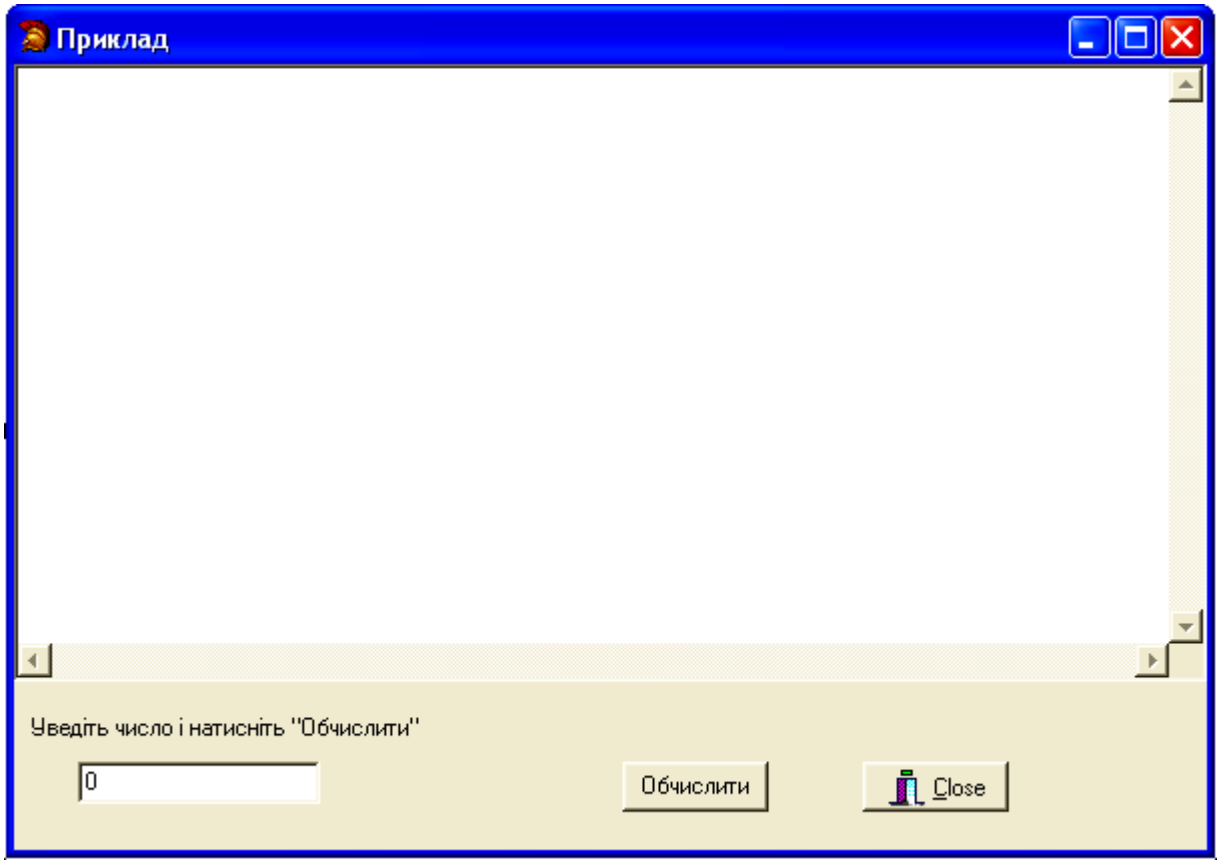


Рис. 2.2 – Можливий остаточний вигляд форми

Оскільки в компонента може бути зовнішня крайка, її досить часто усувають, щоб компонент не виділявся на фоні свого контейнера. Зміна вигляду зовнішньої крайки забезпечується зміною значення властивості BevelOuter. У панелі їй надане значення bvNone – немає крайки.

Властивість Caption (Заголовок) є в багатьох компонентів Delphi і служить для виведення заголовка. Вона змінена у форми – для виведення інформації про номер приклада, у панелі – для знищення заголовка, у міток – для виведення підказки .

Властивість Text (Текст) у поля введення і властивість Lines (Рядки) у багаторядкового поля визначають текст, що буде міститися в однорядковому або багаторядковому полі в момент їхньої появи на екрані. Для компонентів TEdit у властивість Text записане значення 0, щоб навіть за відсутності введеного числа за нього було взяте значення 0. У багаторядкового поля властивість Lines очищена.

Значення bkClose властивості Kind (Сорт) у компонента TBitBtn означає наявність на поверхні кнопки типового значка і напису Close, а також зв'язок з кнопкою функції закриття вікна.

Зміна значень властивостей буде автоматично приводити до зміни тексту модуля. В результаті текст модуля прийме наступний вид:

```
unit Unit1;

interface

uses
  Windows, Messages,SysUtils,Variants,Classes, Graphics,
  Controls, Forms, Dialogs, StdCtrls, Buttons, ExtCtrls;
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Edit11: TEdit;
    Edit12: TEdit;
    Edit13: TEdit;
    Button1: TButton;
    BitBtn1: TBitBtn;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Memo1: TMemo;
    private
      { Private declarations }
    public
      { Public declarations }
  end;

var
  Form1: TForm1;

implementation
{$R *.dfm}

end.
```

Як це видно з наведеного вище тексту модуля, Delphi автоматично вставив у клас розміщені на формі компоненти, надавши їм стандартні імена (якщо компонент при проектуванні форми перейменовувати, змінюючи значення властивості Name, то відповідно буде змінюватися текст модуля).

Етап 2. Написання початкового коду.

Насамперед перейдемо у вікно коду та впишемо у секції **private** опису касу форми такий рядок:

```
a, b, c: Real;
```

Тим самим ми опишемо змінні для позначення коефіцієнтів рівняння, вказавши, що вони приймають дійсні значення.

Виконаємо подвійний клік мишкою над зображенням форми у вікні Дерево Об'єктів або над будь-яким вільним місцем форми в її вікні. У відповідь на клік мишкою Delphi автоматично вставить у код модуля перед термінатором **end** з крапкою такий код:

```
procedure TForm1.FormCreate(Sender: TObject);
begin

end;
```

Це код так званого опрацьовувача події OnCreate (За створенням) для форми.

Слово **procedure** вказує, що опрацьовувач є так званою підпрограмою-процедурою. Складене ім'я `TForm1.FormCreate` – це ім'я процедури, що складається з двох частин: імені ***класу*** `TForm1` і власне імені процедури `FormCreate`. Після імені процедури в круглих дужках зазначено опис параметра, з яким вона буде викликатися (`Sender: TObject`). У нашій програмі параметр `Sender` використовуватися не буде. У більш складних програмах за допомогою цього параметра програміст може визначити, який компонент згенерував дану подію (у цьому випадку подія OnCreate). У загальному випадку у процедури може бути кілька параметрів; вони також можуть бути відсутніми. Перший рядок розглянутого фрагмента коду називається ***заголовком процедури*** (він завершується символом «крапка з комою»).

Слідом за заголовком процедури розташовується її ***тіло*** (у даному випадку воно порожнє й містить тільки так звані операторні дужки **begin** та **end**). Щоб опрацьовувач події виконував які-небудь дії, його тіло повинне бути наповнене операторами. Відзначимо, що відразу ж за заголовком може розміщуватися секція описів, у якій програміст описує допоміжні елементи, необхідні для реалізації тіла процедури.

Крім вставки коду опрацьовувача події, Delphi автоматично модифікує опис класу, вставляючи в нього перед словом **private** заголовок опрацьовувача події:

```
procedure FormCreate (Sender: TObject);
```

Модифікуємо код опрацьовувача події OnCreate форми, надавши йому такого вигляду:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
   DecimalSeparator := '.';
end;
```

Справа в тому, що якщо використовується русифікована версія Windows, то в ній як роздільник цілої та дробової частин дійсного числа використовується не десяткова точка, як це має місце в мовах програмування (у тому числі і в Delphi), а кома. Результатом цього може бути неправильна робота деяких підпрограм, що перетворюють рядки до дійсних чисел (наприклад, `StrToFloat`). Системна змінна `DecimalSeparator` містить символ, що використовується як роздільник цілої та дробової частин дійсних чисел. Єдиний оператор процедури `TForm1.FormCreate` на початку роботи програми змінює цей символ на символ «крапка», що використовується в Delphi та у не русифікованій версії Windows.

Клікнемо тепер мишкою над кнопкою Обчислити. У відповідь на клік мишкою Delphi автоматично вставить у код модуля перед термінатором **end** з крапкою такий код:

```
procedure TForm1.Button1Click(Sender: TObject);
begin

end;
```

Це код так званого опрацьовувача події OnClick (За кліком) для кнопки Обчислити. Взагалі кажучи, при кліку мишкою в працюючій програмі виникає подія OnClick, що зв'язується з компонентом, над яким був здійснений клік.

Якщо опрацьовувач події відсутній, то подія ніяк не опрацьовується; за умови його наявності клік мишкою активізує опрацьовувач і забезпечує виконання дій, записаних у цьому опрацьовувачі.

Крім того, в опис класу перед словом **private** буде автоматично вставлений заголовок опрацьовувача події:

```
procedure Button1Click(Sender: TObject);
```

Модифікуємо код процедури `TForm1.Button1Click`, надавши йому такого вигляду:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  a := StrToFloat(Edit1.Text);
  b := StrToFloat(Edit2.Text);
  a := StrToFloat(Edit3.Text);
  d := b * b - 4 * a * a;
  if d < 0
    Memo1.Lines.Add('The equation has no roots');
    Memo1.Lines.Add('x1 = '
      + FloatToStr((-b + Sqrt(d)) / 2 * a));
    Memo1.Lines.Add('x2 = '
      + FloatToStr((-b - Sqrt(d)) / 2 * a));
end;
```

Етап 3. Виконаємо компіляцію.

Як це видно з рис. 2.3, компілятор виявив 5 синтаксичних помилок 1 натяк та 1 попередження і вивів відповідні пояснення у спеціальне вікно Messages[1)].

Закриємо вікно компіляції, клікнувши мишкою над кнопкою Ok у ньому. Після цього стане активним вікно коду, у якому забарвиться червоним кольором рядок, у якому вперше було знайдено помилку, а курсор розміститься за яким-небудь символом цього рядка. Програміст повинен шукати синтаксичну помилку перед курсором за текстом програми, причому не обов'язково у цьому ж рядку. Якщо здійснити подвійний клік мишкою у вікні Messages над будь-яким з рядків, у вікні коду буде здійснений перехід до рядка, у якому вперше знайдена відповідна помилка.

Попередження і натяк мають такий вигляд:

[DCC Warning] Unit1.pas(47): W1023 Comparing signed and unsigned types - widened both operands – Порівняння знакового та беззнакового операндів – розширюються обидва операнди;

[DCC Hint] Unit1.pas(24): H2219 Private symbol 'c' declared but never used – Приватний символ 'c' оголошений, але ніколи використовуваний.

[1)] Ініормацію про помилки відбито у вікні структури ще до компіляції (див. рис. 2.3).

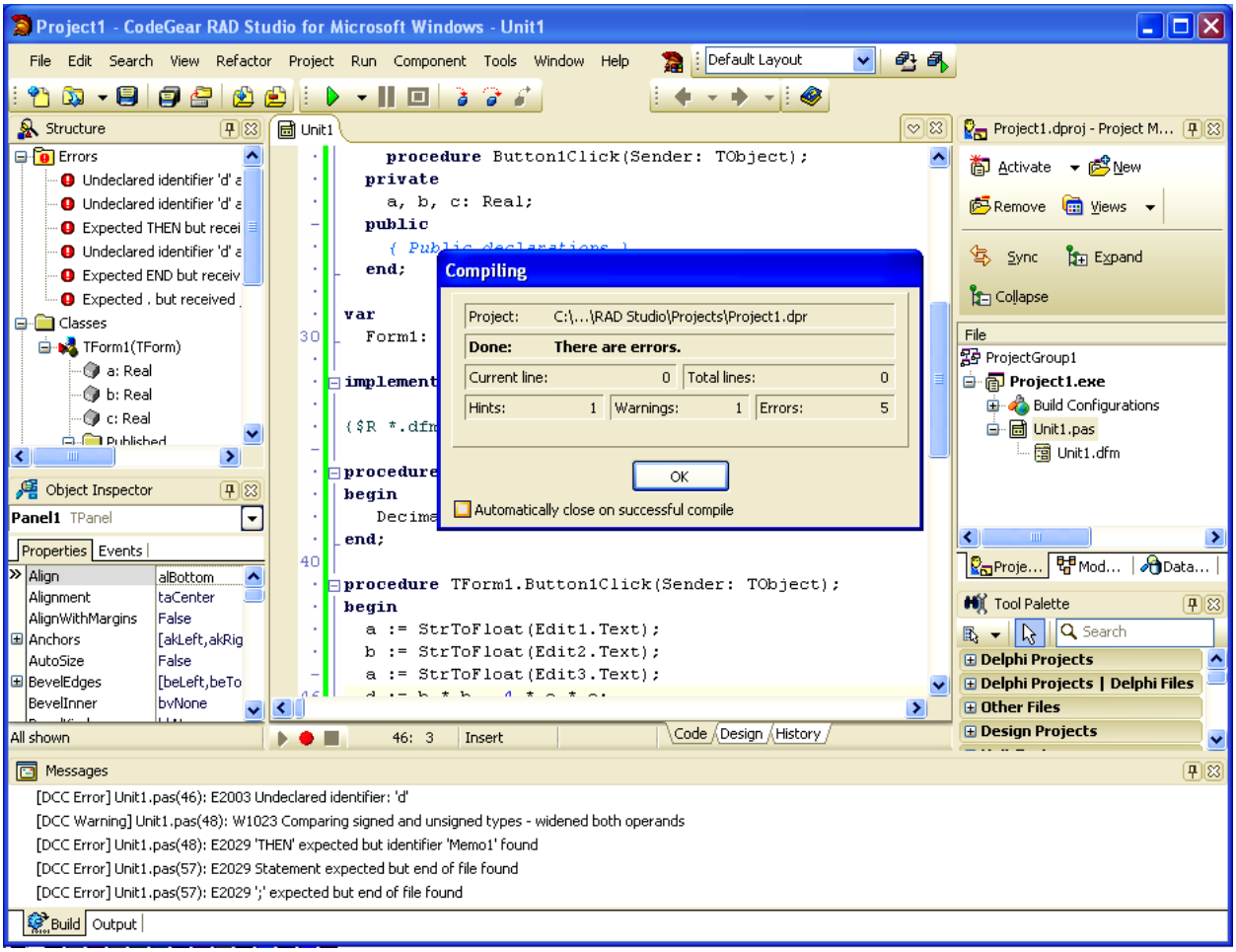


Рис. 2.3 – Результати компіляції

Не будемо поки звертати увагу на натяк та повідомлення і спробуємо зробити виправлення, орієнтуючись тільки на повідомлення про помилки (Error).

У вікні Messages в цьому разі виведена інформація про такі помилки:

[DCC Error] Unit1.pas(46): E2003 Undeclared identifier: 'd' – Невизначений символ 'd';

[DCC Error] Unit1.pas(48): E2029 'THEN' expected but identifier 'Memo1' found – Очікувалося 'THEN', але знайдений ідентифікатор 'Memo1';

[DCC Error] Unit1.pas(57): E2029 Statement expected but end of file found – Очікувався оператор, але знайдений кінець файлу;

[DCC Error] Unit1.pas(57): E2029 ';' expected but end of file found – Очікувалася ';', але знайдений кінець файлу

[DCC Fatal Error] Project1.dpr(5): F2063 Could not compile used unit 'Unit1.pas' – Компілятор не зміг зібрати модуль 'Unit1.pas'

Перше повідомлення говорить про те, що змінна d (для запису значення дискримінанту) використовується, але не визначена у програмі. Тому опишемо її у розділі **implementation** модуля Unit1:

```
var
  d: Real;
```

Друга помилка – це помилка, обумовлена неправильним синтаксисом оператора **if**: в цьому операторі пропущено службове слово **then**. Впишемо це слово після умови `d < 0`.

Дещо складнішими є наступні два повідомлення. Користуючись ними, визначити помилки неможливо, оскільки ці повідомлення обумовлені наявністю інших помилок (тут – раніше розглянутою другою помилкою).

Останнє повідомлення – це узагальнююче повідомлення про наявність синтаксичних помилок.

Таким чином, ми виявили тільки дві помилки.

Виконаємо повторну компіляцію. В результаті отримаємо повідомлення ще про одну помилку:

[DCC Error] Unit1.pas(52): E2029 'END' expected but ')' found – очікувалося 'END', але знайдено ')'.

Якщо клікнути мошкою у вікні Messages над цим повідомленням, то курсор займе місце перед останньою закриваючою дужкою у рядку

```
+ FloatToStr((-b - Sqrt(d)) / 2 * a)));
```

Аналізуючи оператор (а він займає два рядки), виявляємо зайву закриваючу дужку. Знищимо дужку перед символом ';'. Помічаємо також, що у наступному операторі має місце така ж сама помилка. Тому виправимо і цю помилку.

Відзначимо, що компілятор під час першого сеансу своєї роботи не виявив ці помилки. Більш того, під час другого сеансу була виявлена тільки одна помилка, хоч їх було дві.

Етап 4. Натиснемо інструментальну кнопку  , запускаючи програму на виконання, і переконаємось, що компіляція пройде успішно і програма почне виконуватися. Якщо ввести значення $a = 1$, $b = 2$, $c = 1$, то ми переконаємося, що отримані корені, є правильними. Однак для $a = 1$, $b = 5$, $c = 2$ будуть отримані неправильні корені. Тому треба вже шукати логічні помилки.

Етап 5. Відкриємо вікно Watches List, додамо в нього змінну d, помістимо курсор у рядок, наступний за обчисленням значення дискримінанта d, і натиснемо клавішу Ctrl+F10. Після введення вказаних вище значень коефіцієнтів переконуємося, що дискримінант обчислений невірно (у вікні Watches List виведено значення d, що дорівнює 9, замість 17). Помічаємо, що у тексті програми оператор

```
d = b * b - 4 * a * a;
```

неправильний. Виправляємо помилку:

```
d = b * b - 4 * a * c;
```

Якщо виконати ті ж дії, що й раніше, то ми переконаємося, що тепер у вікні Watches List для змінної d буде виведене 0. Таким чином десь вище є логічна помилка, яку треба виправити.

Етап 6. Не перериваючи виконання програми, додамо у вікно Watches List змінні a, b, c (точніше, Form1.a, Form1.b, Form1.c). Бачимо, що змінна c має значення 0, хоч уведення її здійснювалося. У той же час змінна a має значення 2, хоч в неї уводилося значення 1. Для відшукання помилки, припиняємо виконання програми і натискаємо клавішу F10 для нового запуску програми у режимі налаштування. Після декількох натискань клавіші F10 (при цьому будуть по черзі забарвлюватися виконувані рядки) і введення вказаних трьох значень, ми помічаємо, що третій оператор уведення замість того, щоб записати значення 1 у замінну c, записав його у змінну a (це можна було побачити дещо раніше, але людина звичайно бачить ті значення, що явно виділяються – тут значення 0 у змінній c).

Вказану помилку можна було побачити ще після першої компіляції. Річ у тому, що тоді ми не звернули увагу на натяк, в якому вказувалося, що змінна c оголошена, але ніколи не використовується.

Таким чином, ми переконалися, що навіть натяк (Hint) може свідчити про помилку і треба звертати увагу не тільки на повідомлення про помилки, але й на попередження та натяки.

Таким чином, треба виправити оператор

```
a := StrToFloat(Edit3.Text);
```

записавши його так:

```
c := StrToFloat(Edit3.Text);
```

Якщо тепер запустити програму на виконання, то буде отриманий правильний результат, який однак ще не свідчить про відсутність логічних помилок, оскільки ми ще не перевіряли випадок відсутності коренів.

Етап 7. Запустимо програму на виконання і введемо тепер такі значення: $a = 1$, $b = 2$, $c = 3$. У цьому разі ми бачимо, що, з одного боку, програма виводить повідомлення про відсутність коренів, а, з другого боку, вона видає аварійне повідомлення.

Натискаючи декілька разів клавішу F10, бачимо, що оператори виконуються у потрібній послідовності, але після виведення повідомлення про відсутність коренів, здійснюється перехід до рядка, у якому обчислюється та виводиться перший корінь, хоч цей рядок не повинен виконуватися. Справа в тому, що у програмі використаний невірний для цієї задачі формат оператора **if**.

Знищимо перед рядком, де виводиться перший корінь рівняння, символ «крапка з комою» і впишемо такий рядок:

```
else
```

Якщо тепер запустити програму на виконання, то ми побачимо, що помилка залишилася, але тільки по відношенню до другого кореня.

За допомогою клавіші F10 переконуємося у тому, що у програмі виконується оператор, який обчислює другий корінь рівняння, в той час як його виконання не повинне було мати місце. Справа у тому, що у цьому випадку два оператори

```
Memo1.Lines.Add('   x1 = '
  + FloatToStr((-b + Sqrt(d)) / 2 * a));
Memo1.Lines.Add('   x2 = '
  + FloatToStr((-b - Sqrt(d)) / 2 * a));
```

повинні розглядатися як один (так званий складений оператор), для чого їх треба укласти у операторні дужки **begin** – **end**.

Зробимо це і отримаємо такий текст процедури TForm1.Button1Click:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  a := StrToFloat(Edit1.Text);
  b := StrToFloat(Edit2.Text);
  c := StrToFloat(Edit3.Text);
  d := b * b - 4 * a * c;
```

```
  if d < 0 then
    Memo1.Lines.Add('The equation has no roots')
  else begin
    Memo1.Lines.Add('   x1 = '
      + FloatToStr((-b + Sqrt(d)) / 2 * a));
    Memo1.Lines.Add('   x2 = '
      + FloatToStr((-b - Sqrt(d)) / 2 * a));
  end;
end;
```

Запуск програми зі значеннями $a = 1$, $b = 2$, $c = 3$ тепер приведе до отримання правильного результату.

Етап 8. Запустимо програму на виконання і введемо нові значення коефіцієнтів: $a = 2$, $b = 5$, $c = 3$. У цьому разі ми бачимо, що обчислені корені (`x1 = -4`, `x2 = -6`) знову є неправильними (повинні бути отримані значення `x1 = -1`, `x2 = -1.5`). Треба знову здійснювати покрокове виконання програми, натискаючи клавішу F10. Оскільки ми не передбачили змінні для запису коренів рівняння, прийдеться дивитися на результат, що виводиться на екран користувача. В результаті ми побачимо, що всі обчислення до виведення значення першого кореня виконуються правильно, а сам корінь є невірним. Висновок – є помилка у обчисленні кореня. Уважно подивившись на оператор, ми бачимо, що в ньому порушений порядок виконання операцій, а саме, оскільки операції множення та ділення мають один і той же пріоритет, при обчисленні кореня спочатку здійснюється ділення на 2, після чого отриманий результат помножується на a. Щоб отримати правильний результат, необхідно забрати знаменник `2 * a` у дужки, причому це, звісно, потрібно зробити і при обчисленні другого кореня.

Після виправлення тексту отримуємо правильний результат.

Таким чином, одноразове отримання правильного результату не дає гарантії того, що програма буде правильно. Остання помилка не проявлялася раніше, тому що для коефіцієнта a уводилося значення 1, яке не впливало на результат і у разі попадання цього значення у чисельник, і у разі попадання його у знаменник. Це ще раз говорить про необхідність ретельного підбору тестових даних та багаторазового тестування.

3. ЗАВДАННЯ НА ЛАБОРАТОРНУ РОБОТУ

Під час лабораторної роботи потрібно:

1. Повторити описані у розділі 2 дії, пов'язані зі створенням та налаштуванням нового застосунка.

2. Виконати наведені нижче завдання, пов'язані з розробкою форми (див рис. 3.1) та набиранням тексту модуля для нового застосунка, збереженням його, збереженням з новим іменем, завантаженням проекту з диска, редагуванням та налаштуванням програми.

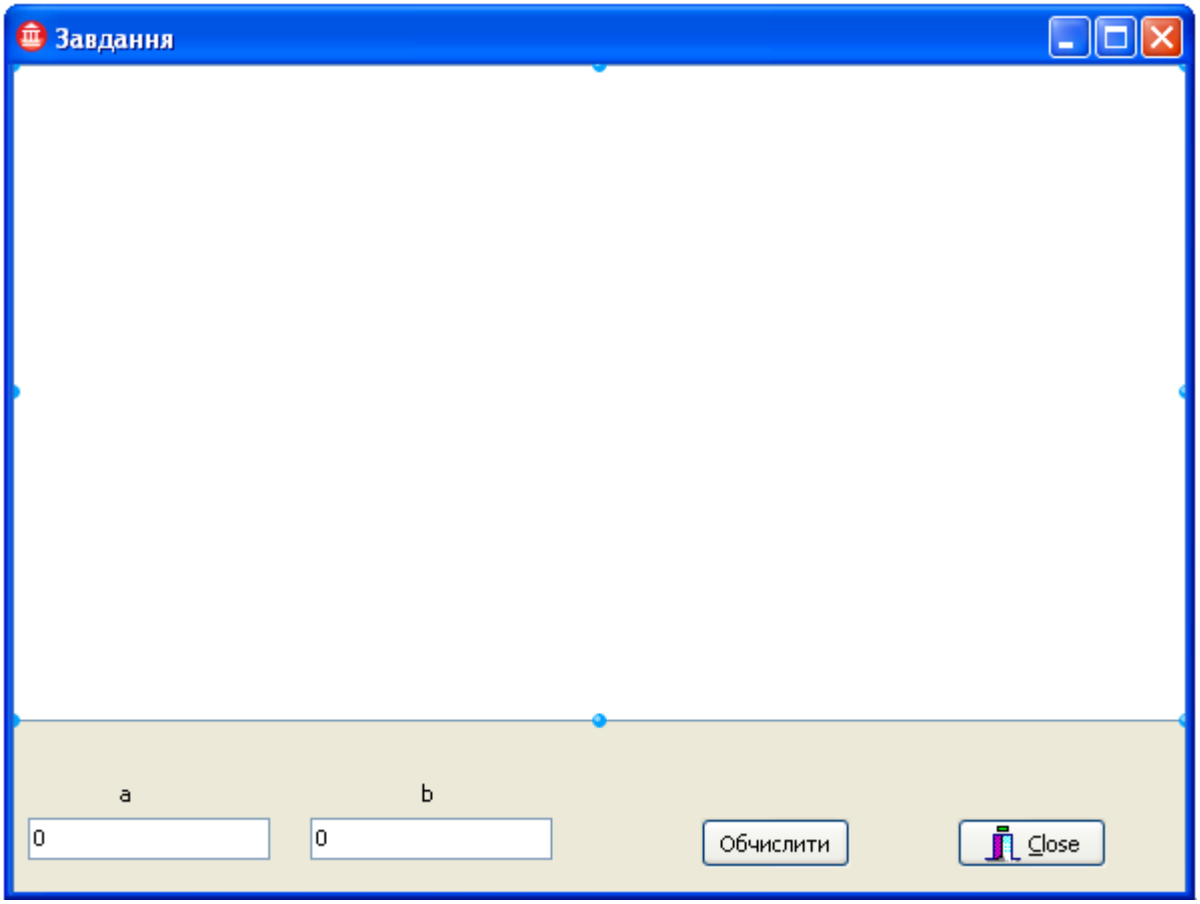


Рис. 3.1 – Вигляд форми для відтворення

Завдання 1.

Створіть застосунок, в якому повторіть наведений нижче текст опрацьовувача події OnClick кнопки Button1. З'ясуйте, що виконує програма.

```
procedure TForm1.Button1Click(Sender: TObject);
var
  a, b, c: Integer;
begin
```

```
  a := StrToInt(Edit1.Text);
  b := StrToInt(Edit2.Text);
  if a > b then
    Memo1.Lines.Add('1 - Yes')
  else
    Memo1.Lines.Add('1 - No');
  c := a * b;
  if c > 100 then
    Memo1.Lines.Add('2 - Yes')
  else
    Memo1.Lines.Add('2 - No');
  if a mod 2 = b mod 2 then
    Memo1.Lines.Add('3 - Yes')
  else
    Memo1.Lines.Add('3 - No');
end;
```

Завдання 2.

Нижче наведено код опрацьовувачем події OnClick кнопки Button1 з синтаксичними помилками. На основі цього коду створіть Windows-застосунок, у якому усуньте всі знайдені синтаксичні помилки.

```
procedure TForm1.Button1Click(Sender: TObject);
var
  a, b, c: Integer;
begin
  a := StrToInt(Edit1.Text);
  b := StrToInt(Edit2.Text);
  if (a >< b) and (b >< 0) then begin
    if a * b > 0 then
      Memo1.Lines.Add('1 - Yes');
    else
      Memo1.Lines.Add('1 - No');
  else
    Memo1.Lines.Add('?');
end;
```

Завдання 3.

Нижче наведено варіант коду, що нормує вектор і здійснює виведення результату. У коді містяться помилки. На основі наведеного коду створіть коректно працюючий застосунок.

```
procedure TForm1.Button1Click(Sender: TObject);
var
  a, b, c, d: Real;
```

```
begin
  a := StrToFloat(Edit1.Text);
  a := StrToFloat(Edit2.Text);
  c = Sqrt(a * a + b * b);
  a = a / d;
  b = b / d;
  Memo1.Lines.Add(FloatToStr(a) + '  ' + FloatToStr(b));
end;
```

4. КОНТРОЛЬНІ ЗАПИТАННЯ

1. Які дії треба виконати для створення нового застосунка?
2. Яке призначення основних вікон інтегрованого середовища Delphi.
3. Яке призначення вікна дизайнера форми?
4. Яке призначення вікна коду?
5. Яке призначення вікна дизайнера форми?
6. Яке призначення менеджера історії?
7. Яке призначення вікна інспектора об’єктів?
8. Яке призначення вікна палітри інструменту?
9. Яке призначення вікна інспектора об’єктів?
10. Як поділяються помилки у програмі? Опишіть їх особливості.
11. Опишіть структуру модуля форми.
12. Що відслідковується при покроковому виконанні програми?
13. У чому полягає відмінність режимів покрокового виконання Trace Into та Step Over?
14. Що таке «виконання до курсору»?
15. Що таке точка переривання і як вона встановлюється та видаляється?
16. У чому відмінність точки переривання і виконання до курсору?
17. Чи можна одночасно встановити декілька точок переривання?
18. Для чого використовується вікно Watches List?
19. Як внести нове ім’я у вікно Watches List?
20. Як видалити ім’я з вікна Watches List?

СПИСОК ЛІТЕРАТУРИ

1. Методичні вказівки до самостійної роботи «Створення та налаштування програм у візуальному середовищі Delphi 2009» з курсу «Програмування» для студентів напряму 6.040302 – Інформатика (спеціалізація «Соціальна інформатика») / М. І. Безменов. – Х. : НТУ «ХПІ», 2011. – 40 с.
2. Безменов, М. І. Основи програмування в середовищі Delphi : навч. посіб. / М. І. Безменов. – Х. : НТУ «ХПІ», 2010. – 608 с.
3. Архангельский, А. Я. Программирование в Delphi 6 / А. Я. Архангельский. – М. : БИНОМ, 2002. – 1120 с.
4. Культин, Н. Б. Основы программирования в Delphi 2007 / Н. Б. Культин. – СПб. : БХВ-Петербург, 2008. – 480 с.

ЗМІСТ

Навчальне видання

Методичні вказівки
до лабораторної роботи
«Створення та налаштування програм у візуальному середовищі Delphi 2009»
з курсу «Програмування» для студентів напряму 6.040302 – Інформатика
(спеціалізація «Соціальна інформатика»)

Укладач БЕЗМЕНОВ Микола Іванович

Відповідальний за випуск О. С. Куценко
Роботу до видання рекомендував О. В. Горелий

За авторською редакцією

План 2011 р., поз. 2 / 78-11

Підп. до друку 23.05.2011 р. Формат 60 × 84 $^{1}/_{16}$. Папір офісний.
Riso-друк. Гарнітура Таймс. Ум. друк. арк. 2,09. Наклад 50 прим.
Зам. № 158. Ціна договірна.

Видавничий центр НТУ «ХПІ».
Свідоцтво про державну реєстрацію ДК № 3657 від 24.12.2009 р.
61002, Харків, вул. Фрунзе, 21

Друкарня НТУ «ХПІ», 61002, Харків, вул. Фрунзе, 21